Expressing OLAP operators with the TAX XML algebra

Marouane Hachicha Hadj Mahboubi Jérôme Darmont
Université de Lyon
(ERIC Lyon 2)
5 av. Pierre Mendès-France
69676 Bron Cedex
France
1st.last@univ-lyon2.fr

ABSTRACT

With the rise of XML as a standard for representing business data, XML data warehouses appear as suitable solutions for Web-based decision-support applications. In this context, it is necessary to allow OLAP analyses over XML data cubes (XOLAP). Thus, XQuery extensions are needed. To help define a formal framework and allow much-needed performance optimizations on analytical queries expressed in XQuery, having an algebra at one's disposal is desirable. However, XOLAP approaches and algebras from the literature still largely rely on the relational model and/or only feature a small number of OLAP operators. In opposition, we propose in this paper to express a broad set of OLAP operators with the TAX XML algebra.

Categories and Subject Descriptors

H.2 [**Database Management**]: Languages

General Terms

XML-OLAP algebras

1. INTRODUCTION

Data warehouses and OLAP (On-Line Analytical Processing) are nowadays technologically mature. However, their complexity makes them unattractive to many potential users, thus DSS (Decision Support System) vendors start developing simple, user-friendly Web-based interfaces. Furthermore, many decision-support applications (e.g., competitive monitoring) require data that are external to the institution exploiting them. In this context, the Web is a tremendous data source and a new trend toward on-line data warehousing is currently emerging, including approaches such as XML warehousing [3, 15, 18].

The XML language is indeed becoming a standard for representing business data [2]. It is also particularly adapted for modeling *complex data* from heterogeneous sources, and particularly the Web. Data we term complex are not only numerical or symbolic, but may be represented in various formats (databases, texts, images, sounds, videos...); diversely structured (relational databases, XML documents...); originating from several different sources; described through several channels or points of view (a video and a text that describe the same meteorological phenomenon, data expressed in different scales or languages...); changing in terms of definition or value (temporal databases, periodical surveys...) [5]. Such data are much easier to logically model and store with XML than with relational models.

Many studies aim at performing OLAP analyses over XML data (XOLAP [16]). A first family mainly rely on the power of relational implementations of OLAP [4, 9, 10, 14]. However, in these approaches, no or very few XML-specific OLAP operators are defined. Most recent studies directly relate to XOLAP [13, 16, 17]. However, each approach focuses on defining one cube operator only. In this paper, we propose to express a broad set of OLAP operators with an existing XML algebra (namely, TAX [8]), so that OLAP analyses can be applied onto native-XML data. On the long run, we are actually aiming at three objectives: (1) contribute to define a formal framework that does not currently exist in the XOLAP context; (2) support the effort for extending the XQuery language to allow OLAP queries, especially with XML-specific operators; (3) allow query optimization for OLAP XQueries. Native-XML DBMSs (Database Management Systems), though in constant progress, are indeed limited in term of performance and would greatly benefit from automatic query optimization, especially for costly analytical queries.

In this paper, we particularly focus on the first objective, i.e., defining XOLAP models. Computational issues shall be addressed later, but they are out of the scope of this paper, which is organized as follows. First, we present the context of this work and our motivation (Section 2). Second, we present the data model supporting our proposal (Section 3). Then, we express OLAP operators in TAX and present three detailed examples of XOLAP operators (Section 4). Finally, we compare our approach to X^3's [17], which is in spirit the most closely related work to ours in the literature (Section 5). We conclude this paper and provide future research directions (Section 6).

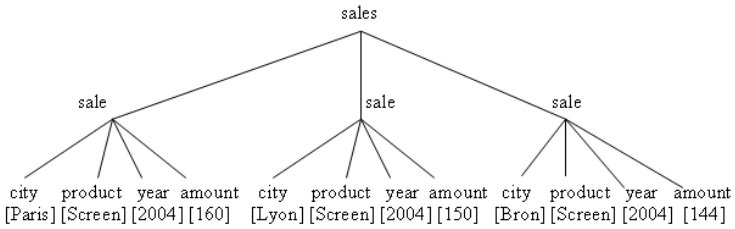


Figure 1: Sample TAX data tree

2. CONTEXT AND MOTIVATION

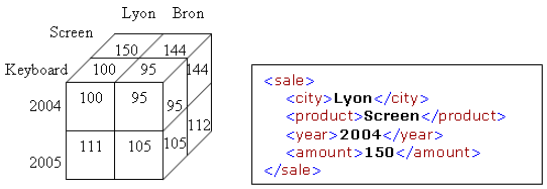
Rather than designing an XOLAP algebra from scratch, we chose to express a selection of "basic" OLAP operators with an existing XML algebra. Many such algebras exist in the literature [6, 7, 11]. Among them, we selected TAX (Tree Algebra for XML [8]) for its richness. TAX indeed includes, under its logical and physical forms, more than twenty operators, which allows us many combinations for expressing XOLAP operators. Furthermore, TAX's expressivity is widely acknowledged. This algebra can be expressed with most XML query languages, and especially XQuery, which we particularly target because of its standard status. Finally, TAX and its derived algebra TLC [12] provide a query optimization framework that we can exploit in the future, since performance is one of our main concerns when desining decision-support applications that are integrally based on XML and XQuery. Unfortunately, due the space constraints, we cannot present the TAX elements that are necessary to understand our XOLAP operators in this paper. The interested reader may refer to the original paper describing TAX [8] for full specifications.

In order to propose a broad range of XOLAP operators, we selected the most common OLAP operators to express them in TAX. When some of them had conflicting definitions, we selected the one that had the broadest acceptation. Before presenting them, we define in Section 3 the data model they are based on, which incorporates multidimensional concepts within the TAX data model. The OLAP operators we selected to express in TAX are structural operators: rotate, switch, push and pull (Section 4.1); set operators: slice and dice (Section 4.2); and granularity-related operators: cube, roll-up and drill-down (Section 4.3). To express these operators, we adapted their formal definition in the classical OLAP context to the XML context by using the possibilities offered by TAX. Each XOLAP operator is actually represented as a combination of TAX operators. In this process, we took care of conserving the formal definitions and properties of OLAP operators, and used the TAX operators exactly as specified by TAX authors.

We define each XOLAP operator formally and, for one of each type (namely, the rotate structural operator, the slice set operator, and the roll-up granularity-related operator), illustrate how it operates on the three levels of the data model: XOLAP cube (conceptual), XOLAP TAX tree (logical) and XML multidimensional document (physical). These multiple representations are aimed at empirically checking whether a given operator proceeds as expected. In Section 4, we use the TAX notations introduced in [8].

3. DATA MODEL

A fact modeled in XML (or XML fact) is, classically, composed of dimension and measure elements. We term a set of XML facts an XML multidimensional document. Since TAX trees can model any XML document, XML multidimensional documents can be represented in TAX, in what we call multidimensional TAX trees, or XOLAP TAX trees. XOLAP TAX trees are thus a subset of TAX trees. An XOLAP TAX tree embeds a collection of fact elements, each described by dimension members and measure values. For instance, Figure 1 features an XOLAP TAX tree that contains *sale* XML facts described by three dimensions: *city* where the sale took place, sold *product*, and sale *year*. Selling *amount* is the measure. Although logically modeled in XML, such facts are still conceptually multidimensional. Thus, we can represent an XML fact by a cube cell. We name such a cube an XOLAP cube, and its cells XOLAP cells. Figure 2(a) shows an example of XOLAP cube. The (physical) XML document corresponding to one of its XOLAP cells is also presented in Figure 2(b).



(a) XOLAP cube (b) XML fact

Figure 2: Sample XOLAP cube and XML fact

In classical OLAP, the concept of order of dimensions is very important. The switch operator's purpose is indeed to manipulate it. Dimensional order is easy to achieve in XML documents and tree representations. In our data model, we thus define a dimension reading direction for XOLAP TAX trees and XML facts: from left to right (in breadth first) and from top to bottom (in depth first), respectively. In addition, in both cases, fact measures must always be placed last in a subtree, with respect to these reading orders (e.g., amount in Figures 1 and 2). This ordering has a very important role in the definition of some XOLAP operators, such as pull and push (Section 4.1).

Finally, our data model takes dimension hierarchies into account, to allow snowflake and constellation-like schemas. Each dimension's hierarchy is represented in a separate TAX tree. An example of such metadata TAX tree is provided

in Figure 3. It refines the sample XOLAP TAX tree from Figure 1's geographical dimension by specifying that cities belong to departments (which are characterized in France by a department number). Note that, though the XOLAP TAX tree from Figure 1 represents only the finest granularity level (city), each city name is a reference to the geographical hierarchy TAX tree. We denote the set constituted of all dimension hierarchy trees $\mathcal{H}$.

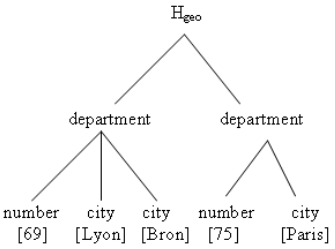


Figure 3: Sample hierarchical dimension specification

4. XOLAP OPERATORS IN TAX

4.1 Structural operators

4.1.1 Rotate

Rotate permutes the dimensions of a cube and simulates its rotation around one of its axes, so as to present different faces of the cube. In other words, rotating helps select cube faces rather than measures. In an XOLAP TAX tree, we must exchange the positions of rotated dimensions for every subtree. Hence, the rotate operator may be expressed in TAX by a selection. Formally, its expression is: $Rotate(C) = \sigma_{P,SL}(C)$, where the rotated dimension is specified in pattern tree P and dimension members and fact measures are specified in selection list SL.

Let us now take the example of rotating the XOLAP cube from Figure 2(a) around the *product* dimension. Figure 4 represents the rotated XOLAP cube (upper left), XOLAP TAX tree (bottom) and XML multidimensional document (upper right). In the selection, dimension order in the input pattern tree indicates the order of the edges in the output tree. Selection list SL must contain the *sale* node to retain all its children in output.

4.1.2 Switch

Switch exchanges the positions of two or several members of a given dimension. Like a rotation, this operation is mainly characterized by a visual effect and preserves measure values when changing the corresponding facts' positions. In an XOLAP TAX tree, we must reorder all subtrees. Hence, the switch operator may simply be expressed in TAX by using the reordering operator: $Switch(C) = \varrho_{P,o,RL}(C)$, where order TVF o and reorder list RL specify the new subtree order.

4.1.3 Push

Push associates the members of a dimension to measures of the cube, i.e., it transforms these members into cell contents. In TAX, push restructures the witness tree. For example, to transform the *city* dimension into a measure, we associate the corresponding edge to the measure. Hence, the edge expressing this measure becomes composed of two subedges: *city* and *amount*. The push operator may thus be expressed in TAX with the copy-and-paste operator: $Push(C) = \kappa_{P,CL,US}(C)$, where copy list CL and update specification US specify how a dimension element is copied into the target fact element.

4.1.4 Pull

Although pull is defined as push's reciprocal operator (transforming a measure into a dimension), it would be a mistake to express it in TAX push's reverse way. Moreover, in an XOLAP TAX tree, a measure is always represented by the last edge of the fact subtree (from left to right). Then, the pull operator just needs to move the measure before other dimensions. In TAX, this may be achieved with the projection operator, the new position being given by projection list PL and pattern tree P: $Pull(C) = \pi_{P,PL}(C)$.

4.2 Set operators

4.2.1 Slice

A data cube may be seen as a set of slices arranged horizontally or vertically. The slice operator dissociates one or several of such slices with respect to one or several attributes from a given dimension. In TAX, we need to select the trees corresponding to these slices. As rotate, the slice operator may be expressed in TAX by the selection operator, but it is also combined to product to attach fact subtrees output by selection under the same root. Moreover, in the rotate operator, the selection operator's pattern tree P indicates dimension order in the output, whereas in the slice operator, it determines the slicing dimension values. Formally, the expression of the slice operator is: $Slice(C) = \times(\sigma_{P,SL}(C))$.

As an example, let us extract, from an initial XOLAP tree, the subtrees corresponding to slices of the corresponding XOLAP cube. Figure 5 shows how the XOLAP cube from Figure 2(a) is sliced on the Keyboard instance of the *product* dimension. It represents the sliced XOLAP cube (upper left), XOLAP TAX tree (bottom) and XML multidimensional document (upper right). Selection pattern tree P determines the slicing dimension member value (Keyboard, here). Selection list SL must contain the *sale* node to retain all its children in output. Finally, the resulting *sale* subtrees must be attached under a common root (*sales*) with the product operator.

4.2.2 Dice

The dice operator extracts a subcube from a data cube with respect to predicates defined on its dimensions. In other words, dicing extracts measures from a cube for a given number of members, which must be identical in all dimensions. In TAX, we may express the dice operator by selecting the corresponding subtrees from the XOLAP TAX tree with a simple pattern tree P and attaching them to the same root with a product: $Dice(C) = \times(\sigma_{P,SL}(C))$. Note that this is again different from the slice operator, in which P specifies the slicing member element(s).

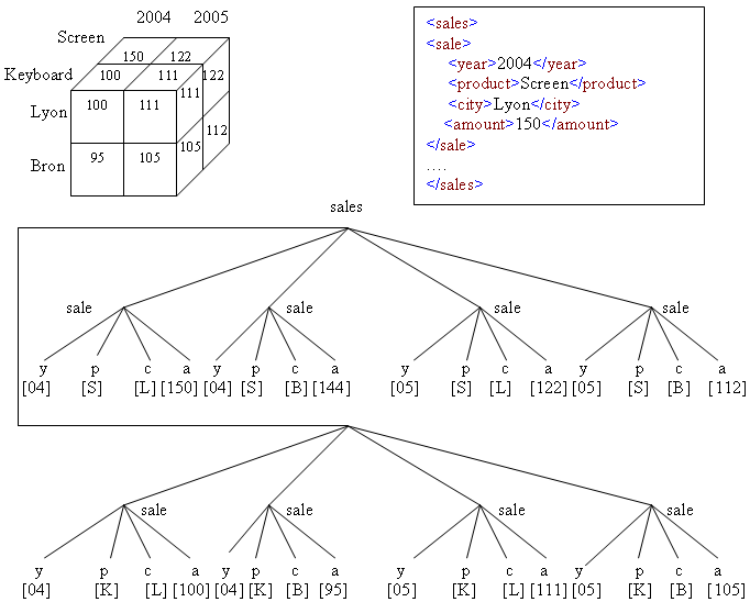


Figure 4: Rotate result example

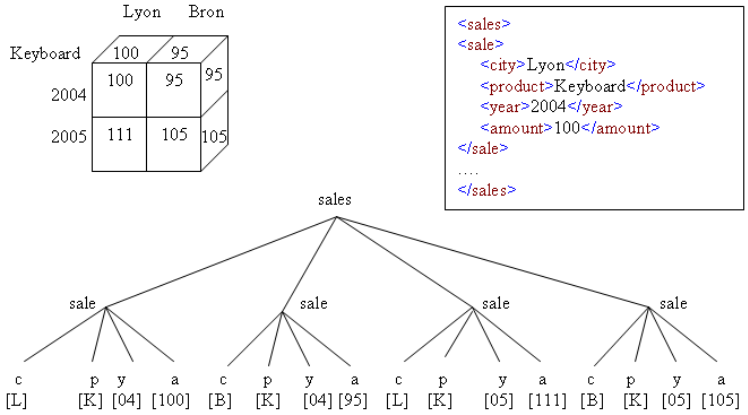


Figure 5: Slice result example

4.3 Granularity-related operators

4.3.1 Roll-up

Roll-up represents a given data cube at a higher (more general) level of granularity, with respect to a dimension hierarchy (e.g., moving from the city granularity to the department granularity along the geographical dimension from Figure 3). During this transition, measures must be aggregated with respect to a grouping operation. In a TAX OLAP tree, this translates as replacing all subtrees with the subtrees resulting from the roll-up operation. The roll-up operator may be expressed in TAX by a combination of the selection, grouping, join, aggregation, node deletion and node insertion operators. Formally, its expression is: $Roll\_up(C, H) = \iota_{P_\iota,IS}(\delta_{P_\delta,DS}(A_{P_A,g,US}(\bowtie_{P_\bowtie,SL_\bowtie}(\gamma_{P_\gamma,g,o}(\sigma_{P_\sigma,SL_\sigma}(C)), H))))$, where $P_\bowtie$ specifies the rolled-up dimension and the selected granularity level, and $H \in \mathcal{H}$ this dimension's hierarchy.

Its execution is illustrated by the following example: perform a roll-up on the *department* dimension (department number 69, here) of the XOLAP TAX tree from Figure 1. Thus, grouping occurs on (*product*, *year*). We use the *sum* aggregation function on *amount*. Figure 6 represents the rolled-up XOLAP cube (upper left), XOLAP TAX tree (bottom) and XML multidimensional document (upper right). The corresponding sequence of TAX operator combination is detailed thereafter.

(1) *Selection:* We must first separate subtrees from the input to prepare them for the next step. (2) *Grouping:* Then, we group the different subtrees by *product* and *year*. Grouping is achieved with respect to all possible combinations of modalities in both dimensions. (3) *Join:* For each tree resulting from grouping, we join subtrees corresponding to the rolled-up dimension (i.e., *city*, here) and to the corresponding dimension hierachical tree H (to retrieve department 69, here). (4) *Aggregation:* Now, for each tree resulting from the previous step, we aggregate all measures (i.e., *amount* here) in one node by using the specified aggregation function (*sum*, here). (5) *Tree updates:* Finally, we must prune

from the output tree the measures at the previous granularity level, which are retained (node deletion) and add a new node describing the resulting granularity level (*department* in our case) with node insertion.

4.3.2 Drill-down

Drill-down is roll-up's reciprocal operator. It helps refine a cube dimension at a lower aggregation level, e.g., moving from the department granularity to the city granularity along the geographical dimension from Figure 3. First, the drilled-down dimension must be selected, then it must be joined to its hierarchy specification $H \in \mathcal{H}$ and the finest-granularity cube C_0 (to retrieve non-aggregate measure values). Then, we must use the projection operator to mask the remaining coarser level member. Finally, measures must be reaggregated to the correct hierarchy level and all subtrees are attached to a single root (product). Formally, the drill-down operator is expressed as follows: $Drill\_down(C, H) = \times(A_{P_A,g,US}(\pi_{P_\pi,PL}(\bowtie_{P_{\bowtie},SL_{\bowtie}} (\sigma_{P_\sigma,SL_\sigma}(C), H, C_0))))$.

4.3.3 Cube

Cube is an aggregation operator that generalizes grouping, roll-up and join on a data cube to produce a more general (aggregate) cube. In TAX, we first must complete a roll-up for each cube dimension (Section 4.3.1). Then, resulting subtrees must be grouped with respect to every possible combination of dimensions. Finally, these subtrees are joined into one tree, with all measures being aggregated. The grouping, join and aggregation TAX operatiors are combined as follows: $Cube(C) = A_{P_A,ag,US}(\bowtie_{P_{\bowtie},SL_{\bowtie}} (A_{P_A,ag,US}(\gamma_{P_\gamma,g,o}(Roll\_up(C, \mathcal{H})))))$.

5. COMPARISON TO X^3

Let us eventually discuss the differences between X^3 [17] and our own XOLAP approach, which have been developed in parallel without knowledge of the other effort. First, we use TAX to support our proposal by expressing each XOLAP operator with one TAX operator or more. On the other hand, Wiwatwattana *et al.* exploit the principle of TAX's grouping operator, but X^3 stands as is and is not based on TAX operators. Thus, it addresses the issues of summarizability and heterogeneity in XML tree structures in an ad-hoc fashion, while our approach is more general.

Another difference between X^3 and our approach lies at data model level. While we exploit TAX's pattern and witness trees, Wiwatwattana *et al.* have based X^3 on relaxed tree patterns, which were initially introduced for approximate XML query matching [1]. Hence, X^3 outputs a lattice whose nodes represent a cuboid, while our result trees are simpler. We indeed do not explicitely take the specificities of the XML model into account and rather trust TAX to do so instead.

Finally, to take on an algorithmic metaphor, X^3's authors have adopted a depth-first approach by fully developping one operator only (which is of little use alone), while we have adopted a breadth-first approach by proposing a wider range of operators (which only apply onto quite "regular" XML data without ragged hierarchies nor missing values, though). Both approaches should aim at completion, in breadth and depth, respectively, to achieve a full XOLAP environment. We actually think they are quite complementary and should be combined, since our XOLAP approach provides modeling/logical functionalities, while X^3 bestows computational support on native XML data.

6. CONCLUSION AND PERSPECTIVES

In this paper, we have completed a first step toward an XOLAP framework, by initiating a previously inexistant formal background (Section 1, objective 1). To achieve this goal, we have demonstrated how the TAX XML algebra could support OLAP operators, by expressing in TAX the main usual OLAP operators (cube, rotate, switch, roll-up, drill-down, slice, dice, pull and push). By doing so, we significantly expanded the number of available XOLAP operators, since up to now, related papers only proposed at most three operators each (always including the cube operator).

The perspectives opened by this work are numerous. The main issue we are currently working on is to adapt the classical OLAP operators we expressed in TAX to actual specificities of XML. More precisely, we are addressing the four following issues, essentially when performing roll-up and drill-down operations: (1) ragged hierarchies [2] in dimensions, i.e., multivalued and multi-granularity dimension members; (2) facts that are defined at heterogeneous levels of granularity (e.g., sales might be detailed at the city level in some countries, while only available at the region level in others); (3) missing (unavailable) dimension information in facts; (4) facts with changing dimensional order (e.g., time and location in one fact, location and time in the next). Our final objective here is to support the XQuery extension effort for decision-support applications (Section 1, objective 2).

Though a non-trivial task that we have postponed for now, we also plan complexity analyses to provide a sounder comparison between the XOLAP operators from the literature and our own proposal. Furthermore, to allow experimentally validating our set of XOLAP operators, we have also started implementing it into a software prototype that helps generate the corresponding XQuery code. This simple Web-based querying interface is currently coupled to the TIMBER XML-native DBMS, but it is actually independent and could operate onto any other DBMS supporting XQuery. Our prototype currently features the rotate, slice and roll-up operators.

This software plaform shall also allow us to initiate a cycle of experimental cost evaluations and optimizations for each XOLAP operator, to find the most efficient expression of each XOLAP operator in TAX (there are, of course, often several solutions for expressing a given XOLAP operator in TAX). In our first experiments, the efficiency of XQuery indeed proved limited when processing complex analytical queries. Hence, we also plan on the long run to exploit our OLAP TAX expression as a basis for automatically optimizing XOLAP queries expressed with XQuery (Section 1, objective 3).

7. ACKNOWLEDGMENTS

The authors would like to thank Dr. Alfredo Cuzzocrea for his thoughtful comments on this paper.

8. REFERENCES

[1] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree Pattern Relaxation. In *8th International Conference on Extending Database Technology (EDBT 02), Prague, Czech Republic*, volume 2287 of *LNCS*, pages 496–513, 2002.

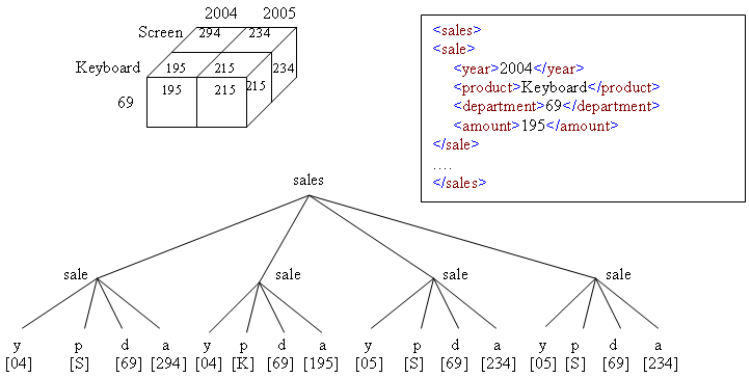


Figure 6: Roll-up result example

[2] K. S. Beyer, D. D. Chamberlin, L. S. Colby, F. Özcan, H. Pirahesh, and Y. Xu. Extending XQuery for Analytics. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 05), Baltimore, USA*, pages 503–514, 2005.

[3] O. Boussaïd, R. BenMessaoud, R. Choquet, and S. Anthoard. X-Warehousing: An XML-Based Approach for Warehousing Complex Data. In *10th East European Conference on Advances in Databases and Information Systems (ADBIS 06), Thessaloniki, Greece*, volume 4152 of *LNCS*, pages 39–54, 2006.

[4] K. Chantola, T. Amagasa, and H. Kitagawa. OLAP Query Processing for XML Data in RDBMS. In *3rd IEEE International Workshop on Databases for Next-Generation Researchers (SWOD 07), In conjunction with ICDE 07, Istanbul, Turkey*, pages 7–12, 2007.

[5] J. Darmont, O. Boussaïd, J.-C. Ralaivao, and K. Aouiche. An Architecture Framework for Complex Data Warehouses. In *7th International Conference on Enterprise Information Systems (ICEIS 05), Miami, USA*, pages 370–373, 2005.

[6] M. F. Fernández, J. Siméon, and P. Wadler. An Algebra for XML Query. In *20th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 00), New Delhi, India*, volume 1974 of *LNCS*, pages 11–45, 2000.

[7] F. Frasincar, G.-J. Houben, and C. Pau. XAL: An Algebra For XML Query Optimization. In *13th Australasian Database Conference (ADC 02), Melbourne, Victoria*, volume 5 of *CRPIT*, 2002.

[8] H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava, and K. Thompson. TAX: A Tree Algebra for XML. In *8th International Workshop on Database Programming Languages (DBPL 01), Frascati, Italy*, volume 2397 of *LNCS*, pages 149–164, 2001.

[9] M. R. Jensen, T. H. Møller, and T. B. Pedersen. Specifying OLAP cubes on XML data. *Journal of Intelligent Information Systems*, 17(2-3):255–280, 2001.

[10] T. Niemi, M. Niinimäki, J. Nummenmaa, and P. Thanisch. Constructing an OLAP cube from distributed XML data. In *5th International Workshop on Data Warehousing and OLAP (DOLAP 02), McLean, USA*, pages 22–27, 2002.

[11] L. Novak and A. V. Zamulin. An XML Algebra for XQuery. In *10th East European Conference on Advances in Databases and Information Systems (ADBIS 06), Thessaloniki, Greece*, volume 4152 of *LNCS*, pages 4–21, 2006.

[12] S. Paparizos, Y. Wu, L. V. S. Lakshmanan, and H. V. Jagadish. Tree Logical Classes for Efficient Evaluation of XQuery. In *SIGMOD International Conference on Management of Data (SIGMOD 04), Paris, France*, pages 71–82, 2004.

[13] B.-K. Park, H. Han, and I.-Y. Song. XML-OLAP: A Multidimensional Analysis Framework for XML Warehouses. In *7th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 05), Copenhagen, Denmark*, volume 3589 of *LNCS*, pages 32–42, 2005.

[14] D. Pedersen, J. Pedersen, and T. B. Pedersen. Integrating XML Data in the TARGIT OLAP System. In *20th International Conference on Data Engineering (ICDE 04), Boston, USA*, pages 778–781, 2004.

[15] J. Pokorný. XML Data Warehouse: Modelling and Querying. In *5th International Baltic Conference (BalticDB&IS 02), Tallin, Estonia*, pages 267–280, 2002.

[16] H. Wang, J. Li, Z. He, and H. Gao. OLAP for XML Data. In *5th International Conference on Computer and Information Technology (CIT 05), Shanghai, China*, pages 233–237, 2005.

[17] N. Wiwatwattana, H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. X^3: A Cube Operator for XML OLAP. In *23rd International Conference on Data Engineering (ICDE 07), Istanbul, Turkey*, pages 916–925, 2007.

[18] J. Zhang, W. Wang, H. Liu, and S. Zhang. X-warehouse: building query pattern-driven data. In *14th international conference on World Wide Web (WWW 05), Chiba, Japan*, pages 896–897, 2005.